Consommation et
Affaires commerciales Canada

Bureau des brevets

Ottawa, Canada
K1A 0C9

Consumer and
Corporate Affairs Canada

Patent Office

(21) (A1) 2,097,099

(22) 1993/05/27

(43) 1994/03/24

5,077,0/87

(51) INTL.CL.^{5} G07F-017/24

(19) (CA) **APPLICATION FOR CANADIAN PATENT** (12)

(54) Method for Field Programming an Electronic Parking Meter

(72) Church, Donald W. - Canada ;
Martin, Scott R. - Canada ;
McGuirk, Kenneth J. - Canada ;
Doucette, John D. - Canada ;
Vallée, Richard E. - Canada ;

(71) Same as inventor

(30) (US) 07/950,097 1992/09/23

(57) 1 Claim

Notice: This application is as filed and may therefore contain an incomplete specification.

Canada

CCA 3254 (10-92) 41 7530-21-936-3254

2097099

ABSTRACT

An electronic parking meter is programmable in the field, in order to provide flexibility and adaptability to future conditions, by partitioning its software program into two independent modules, the smaller of which controls the replacement by a new program module of the other module or of itself.

WHAT IS CLAIMED IS:

1. A method for field programming an electronic parking meter, comprising:

(a) controlling data processing means of said parking meter by means of separate first and second program modules;

(b) providing a predetermined interrupt signal to said data processing means;

(c) causing said data processing means to request a third program module in response to said predetermined interrupt;

(d) said first program module causing said data processing means to store said third program module; and

(e) causing said data processing means to replace one of said first and second program modules with said third program module.

METHOD FOR FIELD PROGRAMMING AN ELECTRONIC PARKING METER

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to parking meters in general, and in particular to electronic parking meters. More particularly still, it relates to field programmable, and reprogrammable, parking meters.

2. Prior Art of the Invention

United Kingdom Patent application GB 2 077 475 published December 16, 1981 discloses a vehicle parking meter which differs from previous mechanically operated meters in that the coin registration, timing, and display functions are performed wholly by electronic circuitry. Preferred form of display is of the liquid crystal type.

The power consumption of the apparatus is very low as it consists predominantly of CMOS circuitry and power is provided by a battery whose charge is maintained by light-activated solar cells.

Functions additional to those provided by mechanical meters are provided and include cash totalization, cash display, settable parking charge per hour, settable parking periods, and provision for providing digital information from the meter to an external data-recording device.

Embodiments are described incorporating the RCA 1802 and RCA 1804 microprocessor together with peripheral circuitry.

United States patent number 4,823,928 issued April 25, 1989 to Speas discloses an electronic parking meter system for receiving at least one type of coin or other payment device and having an electronic parking meter and an auditor. The electronic parking meter comprises a power source which may be a solar type power source, as well as, having terminals for connection to an external source of power. The meter also has a microprocessor with a memory connected to the power supply. An electronic display is connected to the microprocessor and displays pertinent information for the meter. The auditor may be connected to the

microprocessor in the electronic meter by means of a direct cable link or by infrared transmission. The electronic parking meter system may have a sonar range finder connected to the microprocessor in the meter which detects the presence or absence of a vehicle in an associated parking space with the parking meter.

Both of the above prior art documents are incorporated herein by reference.

SUMMARY OF THE INVENTION

The present invention endeavors to provide a flexible, software controlled, parking meter. In order to be flexible, the meter must be able to accommodate changes in the coins it will accept after it has been in use in the field. It is also advantageous to be able to update or correct software "bugs" in the field. It is, therefore, a feature of the present parking meter that software can be "downloaded" into it, preferably by wireless data communication, for example by means of intra-red (IR) receive and transmit channels.

Accordingly, the present invention provides a method for field programming an electronic parking meter comprising: (a) controlling data processing means of said parking meter by means of separate first and second program modules; (b) providing a predetermined interrupt signal to said data processing means; (c) causing said data processing means to request a third program module in response to said predetermined interrupt; (d) said first program module causing said data processing means to store said third program module; and (e) causing said data processing means to replace one of said first and second program modules with said third program module.

BRIEF DESCRIPTION OF THE DRAWINGS

The preferred embodiment of the present invention will now be described in conjunction with annexed drawings, in which:

Figure 1 is an overall block diagram of the parking meter of the present invention;

Figure 2 is a block diagram of the block labelled ASIC in Figure 1;

Figure 3 is a block diagram of the communication interface of Figure 2;

Figure 4 is a high level flow-chart of the overall software of the parking meter;

Figure 5 is a flow-chart of the block labelled "Service IR Interrupt" in Figure 4; and

Figure 6 is a flow-chart of the block labelled "Download Software" in Figure 4.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

Referring to Figure 1 of the drawings, the parking meter comprises an application specific integrated circuit (ASIC) 10, communicating via DATA and ADRESS buses with a central processor (CPU) 11, a programmable memory (EEPROM) 12, and a random access memory (RAM) 13.

The ASIC 10 also receives inputs from a coin shute 14 re position, size and mass of a coin passing through; and receives and transmits serial data SDIN and SDOUT, respectively, from infra-red receiver (IR RCV) 15 and infra-red transmitter (IR TX) 16. A voltage regulator and controller (REG) 17 generates voltages V_{AA} and V_{DD} from battery voltage V_{CC}. V_{AA} is necessary for powering the ASIC 10, while V_{DD} powers the CPU 11, EEPROM 12 and RAM 13. V_{CC} directly powers the IR transmitter 16 as well as red and yellow LEDs 18 and 19. The ASIC 10 drives the liquid crystal displays (front and back) LCD0 and LCD1.

Figure 2 shows a block diagram of the ASIC 10, which comprises an address bus interface 20, a time-base clock 21 controlled by a 3.58 MHz crystal, a real-time clock (RTC) 22 controlled by a micro-power (watch) 32.768 KHz crystal, an LCD display driver 23, a programmable In/Out bus 24, a local RAM 25, a CPU interrupt controller 26, an event counter 27, a coin discrimination interface 28, a universal asynchronous receiver/transmitter (UART) 29, and an IR communications interface 30. All of the above are conventional units,

whose functions could also be implemented by means of software but are more economically implemented by means of an ASIC.

Figure 3 illustrates the interface of the IR communications interface 30 with other components. It comprises a modulator 31 and demodulator 32, a base-band selector 33, and a modulation detector 34, (which is strobed by an 8 millisecond window from the RTC 22 at a rate selectable from 1, 2, 8 or 32 Hz, in order to save power).

Figure 4 shows a flow chart of the overall system software, while Figures 5 and 6 show details of the blocks labelled "Service IR Interrupt" and "Download Software", respectively. In order to be able to field program the parking meters, the software has been divided into two distinct units: the "bootstrap software" and the "application software". Both may be replaced in the field under the control of the existing bootstrap software. Thus for purposes of the present invention the bootstrap software is the essential unit. The application software serves as the interface between

the electronic parking meter (EPM) and the user, and should, therefore, be written in a high level language (such as C) in order to be easily altered to suit differing applications. The bootstrap software, on the other hand, is a low-level program which serves as the interface between the EPM hardware and the application software.

The bootstrap is always the first program to run whenever the EPM goes from "sleep" into operational mode. In general, the primary job of the bootstrap is to quickly jump to the application software if it is present, however, if the application is not present, then the bootstrap will attempt to perform a download of application software.

When the bootstrap is invoked, it first initializes the stack pointer and tests a SLEEP indicator bit in the CPU STATUS CONTROL register to determine if the cause of the wake up was due to an interrupt (SLEEP bit is set) or to a manual EPM reset (SLEEP bit is clear). If the SLEEP bit is set, the bootstrap tests the application version byte for zero or non-zero to determine if

application software is present in the EPM. A non-zero version byte will cause the bootstrap to enter the application software, otherwise, the bootstrap assumes control of the EPM. If the SLEEP bit was clear on entry to the bootstrap, the bootstrap initializes the EPM I/O ports and serial port, and assumes control of the EPM.

When the bootstrap assumes control of the EPM, it immediately loads a software download utility into RAM and enters it. Placing the download utility into RAM permits downloading of either bootstrap or application into the EEPROM since software is not executing from EEPROM. The download utility will attempt to download new EPM software (either bootstrap or application) provided a remote terminal is requesting communications with the EPM. Once software is loaded or the communications link is removed the download utility will exit back to the bootstrap. The bootstrap will then retest the application version byte and enter the application if the version byte is non-zero. Otherwise, the bootstrap will enable automatic serial port sampling, display "E001" and enter sleep mode.

The process of downloading software requires the use of two basic communication functions: put packet() and get packet(). Both functions transfer data through the serial port in a consistent format to be referred to as a packet. Details on packet format are described in Table 1 below. The put packet() routine assembles and transmits packets of data based on the length and address of the data field passed to it by the calling function. The get packet() routine polls the serial receiver looking for a valid packet of data. It will poll the receiver until it receives a valid packet, an erroneous packet, looses the communication link or times out. If a valid packet of data was received, get packed() will use a pointer passed by the calling function to store the data. The calling function will receive status information when get packet() returns to determine if it was successful.

Table 1 - Packet Format

Byte Number	Content
0	SYNC
1	STX
2	Packet Length (LSB)
3	Packet Length (MSB)
4	Packet Type
5	Packet Sequence
6	Start of Data Field
N	End of Data Field
N+1	Checksum
N+2	ETX

Notes:

1. Packet length and checksum includes bytes 4 to N.
2. Packet types are either BOOTSTRAP or APPLICATION.
3. Packet sequences always starts at zero.

The EPM always show "dddd" on its LCD display while the download utility is operating. The general philosophy of the communications between a remote terminal and the EPM is that the EPM software always initiates data transfers. Therefore, the EPM download utility starts a software download procedure by transmitting "request-for-software" control packets on a regular basis and waiting for a response. The remote

terminal responds with a "software-initiation" packet that contains information such as software type, start address and the length (in packets and in bytes) of the software to be downloaded. The download utility extracts this information and then asks for each packet of software in succession. As each software packet is being received the data is temporarily stored in RAM so it can be sumchecked before committing it to EEPROM. If the sumcheck fails the packet, it will be requested again. During this time, the EPM will display "ddxx" where "xx" is the number of packets left to be downloaded. After all packets are received, the download utility will exit back to the bootstrap.

If the software to be downloaded is bootstrap, the number of software packets for the download must be one. This is done to prevent the possibility of a communications link disruption from leaving a partially loaded bootstrap in the EPM. As a result, the entire bootstrap code (1024 bytes or less) will be safely loaded into the EPM RAM before updating is carried out. It should also be noted that new bootstrap will void any application that may have been present in the EPM.

Application software can be downloaded with a variable number of packets of variable length. While each packet is sumchecked before it is written to EEPROM, there is a final sumcheck performed on the entire application code in the EEPROM after download is complete. If the code is verified, the download utility will update the application version byte and return to the bootstrap, otherwise, it will restart the download procedure. Should the communications link be removed any time after a download is started, the download utility will clear the application version byte and exit back to the bootstrap.

The bootstrap code which includes the utilities discussed above along with several support functions occupies not more than the first 1024 bytes of the EEPROM. The functions which have been included in the bootstrap are shared by the bootstrap and application. Independence of bootstrap and application is maintained by requiring the application to use a jump table located in the bootstrap to use the bootstrap functions. All shared functions in the bootstrap have been written so that they abide by standard "C" calling conventions:

disable watchdog code()	- disables watchdog timer circuit
getpacket code(timeout, *buffer)	- gets a packet from the serial port
go to sleep code (interrupt mask)	- puts the EPM into sleep mode
initialize uart code()	- initializes the UART
lcdputhex code(hexval)	- displays "hexval" in hex on the EPM
memcpy code(*source, *dest, length)	- copies "length" bytes from src to dest
putpacket code(packet len, *buffer)	- transmits a packet out the serial port
reset watchdog code()	- enable watchdog timer circuit

A full pseudo code source listing for the bootstrap software is given in the following fourteen pages.

A suitable hand-held device for wireless (intra-red) communication with the parking meter is comprised of a PSION ORGANISER II, made by Psion (Psionhouse, Harcourt Street, London W1H 1DT, England) together with a EXTECH IR COMMS LINK (Part Numbers 767321, 767322, 767324) made by Extech Instruments Corporation (335 Bear Hill Road, Waltham MA 02154).

```
File BS :

Routine download_software :

  Load interrupt mask with correct bits
  Call lcdputhex     to display download status

  Label ask_for_control_packet :

  Load request packet with data
  Call request_and_receive_packet

  If return value was negative
    - Jump to ask_for_control_packet

  Else if return value was 0
    - Jump exit_no_download

  Else if received packet not correct length
    - Jump to ask_for_control_packet

  Else if received packet not correct type
    - Jump to ask_for_control_packet

  Else if received packet not correct sequence
    - Jump to ask_for_control_packet

  Else if going to receive bootstrap and not just 1 block
    - Jump to ask_for_control_packet

  Else if battery low
    - Jump to ask_for_control_packet

  Label set_up_for_download:

  Setup pointer to EEPROM location
  Set packet counter to one
  Save current packet sequence

  Label ask_for_next_data_packet :

  Call lcdputhex     to display status
  Call request_and_receive_packet

  If return value negative
    - Jump ask_for_next_data_packet

  Else if return value zero
    - Jump exit_download_fail

  Else if not correct packet type
    - Jump ask_for_control_packet

  Else if not correct sequence
    - Jump ask_for_next_packet

  Increment packet counter
```

```
Label received_next_data_packet :

Setup number of bytes and memory location to read & write
Call do_eeprom_write     to copy data to EEPROM

If still more packets
  - Jump ask_for_next_data_packet

If downloaded bootstrap code
  - Jump bs_download_ok

Setup length, addresses and checksum
Call verify_eeprom

If checksum not zero
  - Jump ask_for_control_packet

Label write_version :

Setup memory address
Call do_eeprom_write     to write data to eeprom
Clear sleep bit

Label exit_no_download :

Return to caller

Label exit_download_fail :

Clear version information
Jump write_version

Label bs_download_ok :

Jump to address 0000H

{---------------------------------------------------------------}

Routine do_eeprom_write :

  Set eeprom write enable line
  Calculate page boundary
  If number of bytes < 256
    - Jump check_lsb

  Label go_to_write_page:

  Load number of bytes to page boundary
  Jump write_page

  Label check_lsb:

  If number of bytes => page boundary
    - Jump got_to_write_page

  Else
    - Jump write_page

{---------------------------------------------------------------}
```

```
Routine page_mode :

  If bytes > 255
    - Jump max_page_write

  Else if bytes = 0
    - Jump exit_page_mode

  Else if bytes < 32
    - Jump write_page

  Label max_page_write:

  Set bytes to write to 32

  Label write_page :

  If not finished
    - Jump write_page

  Label wait_for_write_cycle :

  If bit 7 at both source and destination not the same
    - Jump wait_for_write_cycle

  Else
    - Decrement counters

  Jump page_mode

  Label exit_page_mode:

  Return

{-----------------------------------------------------------------}

Routine bootstrap :

  Setup start of ram
  If sleep bit clear
    - Jump bootstrap_control

  Label epm_software_exists_? :

  If Call check_version..   returns zero
    - Jump go_get_software

  Label jump_to_software:

  Jump to start of software (CSTARTUP in application listing)

{-----------------------------------------------------------------}
```

```
Routine bootstrap_control:

  Turn on LCD
  Setup IO data directions
  Call initialize_uart_code
  Delay for hardware

  Label go_get_software :

  Call init_ram_funcs
  Call download_software

  If Call check_version    returns non-zero
    - Jump jump_to_software

  Label bootstrap_ok :

  Call lcdputhex_code     to indicate status

  Label bootstrap_sleep :

  Call go_to_sleep_code

{----------------------------------------------------------}

Routine check_version :

  Compare version in eeprom with passed parameter
  Return with sero flag status

{----------------------------------------------------------}

Routine init_ram_funcs :

  Initialize RAM routines and variables.
  Copy functions starting at 0000 to ram_func_end to ram.
  Return to caller

{----------------------------------------------------------}

Routine request_and_receive_packet :

  Save address of packet we want to get.
  Load number of data bytes in request packet.
  Call putpacket_code
  Retreive address of packet to get.

  If Call getpacket_code    for next packet not successful
    - Jump exit_no_kick

  Else
    - Call reset_watchdog_code

  Label exit_no_kick:

  Return to caller

{----------------------------------------------------------}
```

```
Routine verify_eeprom :

  Save parameter checksum
  Calculate checksum from ram locations
  Set carry flag by comparing
  Return to caller

{--------------------------------------------------------------}

Address ram_func_buffer :
   declare 200H bytes

Address request_packet :

Address request_type :
   declare 1 byte

Address request_sequence :
   declare 1 byte

Address request_data :
   declare 17 byte

Address ctrl_type :
   declare 1 byte

Address ctrl_sequence:
   declare 1 byte

Address dnld_start :
   declare 2 bytes

Address version :
   declare 4 bytes

Address dnld_length :
   declare 2 bytes

Address chksum :
   declare 1 byte

Address dnld_type :
   declare 1 byte

Address num_packetsv:
   declare 1 byte

Address data_packet :

Address data_type :
   declare 1 byte

Address data_sequence :
   declare 1 byte

Address data :
   declare MAX_PACKET_LEN bytes

{--------------------------------------------------------------}
```

```
File GET :

Routine g tpacket_code :

  Label get_SYNC:

  Turn on receive data enable bit.
  Turn on baud clock.

  If Call check_abort      indicates abort
    - Jump exit_lost_link

  If Call packet_getc      gets no character
    - Jump exit_timeout

  Else if character is not what's expected
    - Jump get_SYNC

  If Call check_abort      indicates abort
    - Jump exit_lost_link

  If Call packet_getc      gets no character
    - Jump exit_timeout

  Else if character is not what's expected
    - Jump get_SYNC

  If Call check_abort      indicates abort
    - Jump exit_lost_link

  If Call packet_getc      gets no character
    - Jump exit_timeout

  Save character

  If Call check_abort      indicates abort
    - Jump exit_lost_link

  If Call packet_getc      gets no character
    - Jump exit_timeout

  Save character, now have length of expected data
  Setup pointers to where to store the data
  Initialize checksum value

  Label get_next_byte :

  If Call check_abort      indicates abort
    - Jump exit_lost_link

  If Call packet_getc      gets no character
    - Jump exit_timeout

  Update pointer to next location
  Add character to checksum

  If more data left to get
    - Jump get_next_byte
```

```
Label get_check_sum :

If Call check_abort( )   indicates abort
  - Jump exit_lost_link

If Call packet_getc( )   gets no character
  - Jump exit_timeout

Save checksum we just got

If Call check_abort( )   indicates abort
  - Jump exit_lost_link

If Call packet_getc( )   gets no character
  - Jump exit_timeout

Else if not expected character
  - Jump get_sync

If calculated checksum and received checksum not the same
  - Jump exit_bad_chksum

Jump exit_return

Label exit_lost_link :

Set status
Jump exit_return

Label exit_timeout :

Set IO2 to output
Drive IR enable from IO2
Drive IR enable high.
Delay for hardware
Drive IR enable low.
Drive IR enable from ASIC RXE line.
Set status
Jump exit_return

Label exit_bad_chksum :

Set status

Label exit_return :

Turn off receive data enable bit.
Return to caller

{------------------------------------------------------------}
```

```
Routine pack t_getc :

  Label check_for_data :

  If no data in receive buffer
    - Jump check_for_timeout

  Save new character
  Clear carry flag for success
  Jump exit

  Label check_for_timeout :

  If Call check_abort() indicates abort
    - Jump exit

  If not timed out yet
    Jump check_for_data

  Set carry flag for failure

  Label exit:

  Return to caller

{--------------------------------------------------------------}
```

```
File MEMCPY :

Routine memcpy_code :

  Clear carry flag
  Get address of last BS code space byte
  Subtract destination from it

  If illegal destination
    - Jump exit_failure

  Call init_ram_funcs()
  Call do_eeprom_write()

  Label exit_success :

  Load register with success for caller
  Return to caller

  Label exit_failure:
  Load register with failure for caller
  Return to caller

{-------------------------------------------------------------}
```

```
File PUT :

Routine putpacket_code:

  Turn on baud clock.

  If call packet_putc( ) to send first character of packet fails
    - Jump exit_lost_link

  If call packet_putc( ) to send second character fails
    - Jump exit_lost_link

  If call packet_putc( ) to send third character fails
    - Jump exit_lost_link

  If call packet_putc( ) to send fourth character fails
    - Jump exit_lost_link

  Initalize checksum value
  Start at first character to transmit

  Label while_buf_not_empty :

  If no characters left to do
    - Jump exit

  Get next character from buffer

  If call packet_putc( ) to send character fails
    - Jump exit_lost_link

  Add to current checksum

  Jump while_buf_not_empty

  Label exit :

  If call packet_putc( ) to send checksum fails
    - Jump exit_lost_link

  If call packet_putc( ) to send last character fails
    - Jump exit_lost_link

  Load register to indicate success
  Return to caller

  Label exit_lost_link :

  Load register to indicate failure
  Return to caller

{-------------------------------------------------------------}
```

```
Routine packet_putc :

  Delay for hardwar

  Label buf_not_empty :

  If call check_abort     indicates abort
    - Jump putc_exit

  Get status of serial register.
  Test transmit buffer empty.
  Wait for buffer to clear (previous char).
  Write char to transmit buffer

  Label putc_exit :
  Return to caller

{-------------------------------------------------------------}

Routine check_abort :

  If coin_interrupt or clock_interrupt or Not IR interrupt
    - Return abort to caller

  Return continue to caller

{-------------------------------------------------------------}
```

```
File LCDHEX :

Routine lcdputhex_code :

  Retrieve digits from parameter
  Point to first LCD register
  Setup for first nibble

  Label next_hex_digit :

  Retrieve current nibble (working left to right) from parameter
  Point to LCD character to display.
  Get LCD character to display from lookup table.
  Display it at the current LCD digit.
  Point to next LCD display digit.
  Increment current nibble

  If nibble < 5
    - Jump next_hex_digit

   Return to caller

{------------------------------------------------------------}

Address _lcdhexchars :

  Define variables for each digit 0..F

{------------------------------------------------------------}

File SLEEP :

Routine go_to_sleep_code :

  Call initialize_uart_code
  Mask on the correct interrupts
  Sleep with IR on, baud clock off.
  Make RXE the input to IO2
  Clear the sleep bit
  Wait a bit for system
  Halt system

{------------------------------------------------------------}

File UART :

Routine initialize_uart_code :

  Setup uart with correct baud, parity etc.
  Output character to uart's transmit register to initialize
  Delay for hardware to send start bit
  Return to caller

{------------------------------------------------------------}
```

```
File WATCHDOG :

Routine disabl _watchdog_cod  :

  Get curr nt byte value.
  Set watchdog bits to disable value.
  Disable watchdog.
  Return to caller

{-------------------------------------------------------------}

Routine reset_watchdog_code :

  Call disable_watchdog_code(17)
  Get current byte value.
  Set watchdog bits to enable value.
  Enable watchdog
  Return to caller

{-------------------------------------------------------------}
```

2097099

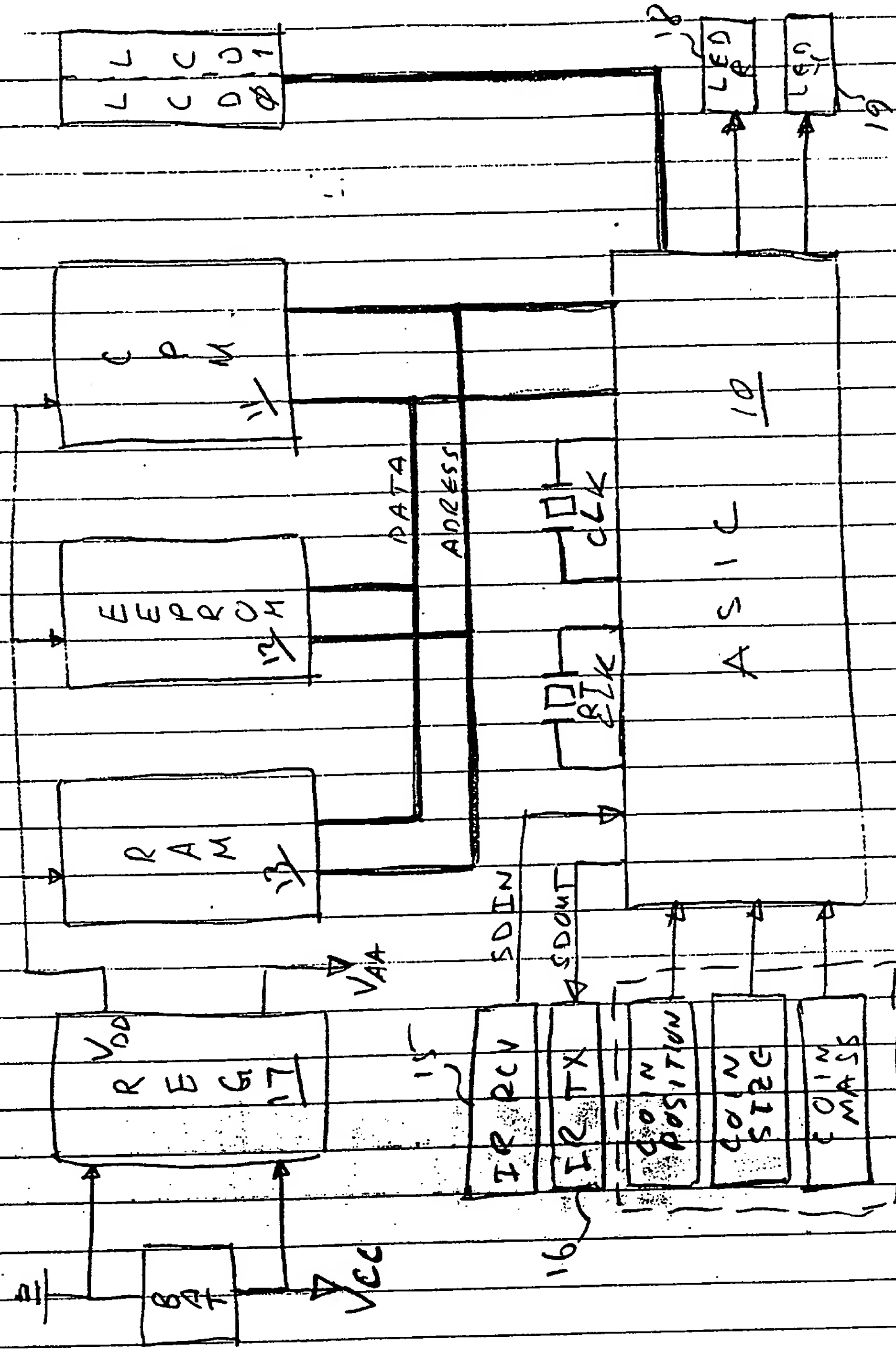

FIG. 1

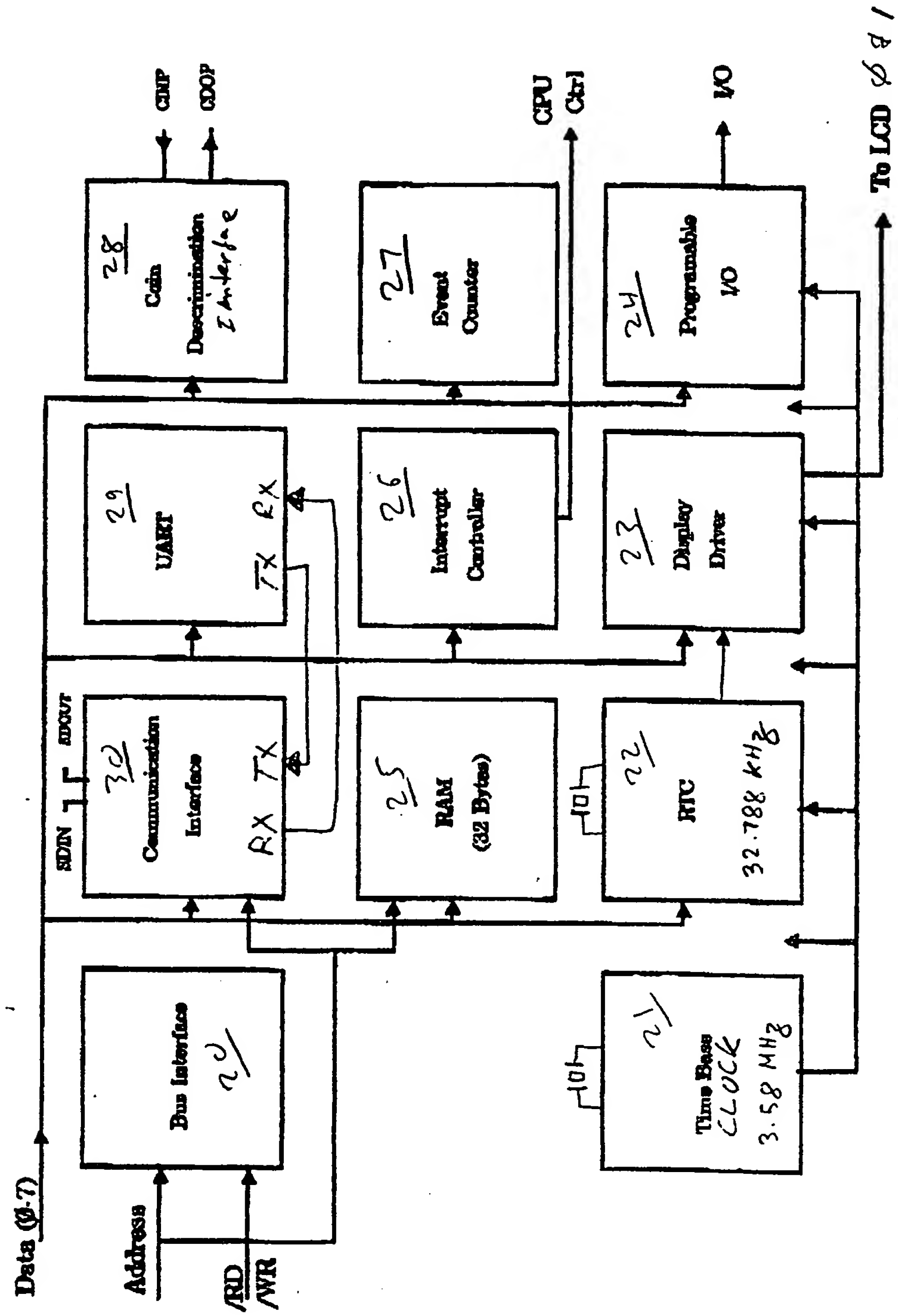
Data (Ø-7)
Address
/RD
/WR
20 Bus Interface
21 Time Base CLOCK 3.58 MHz
22 RTC 32.788 KHz
23 Display Driver
24 Programmable I/O
25 RAM (32 Bytes)
26 Interrupt Controller
27 Event Counter
28 Coin Discrimination Interface
29 UART
TX
RX
30 Communication Interface
SDIN
SDOUT
RX
TX
CINP
CDOP
CPU Ctrl
I/O
To LCD

FIGURE 2

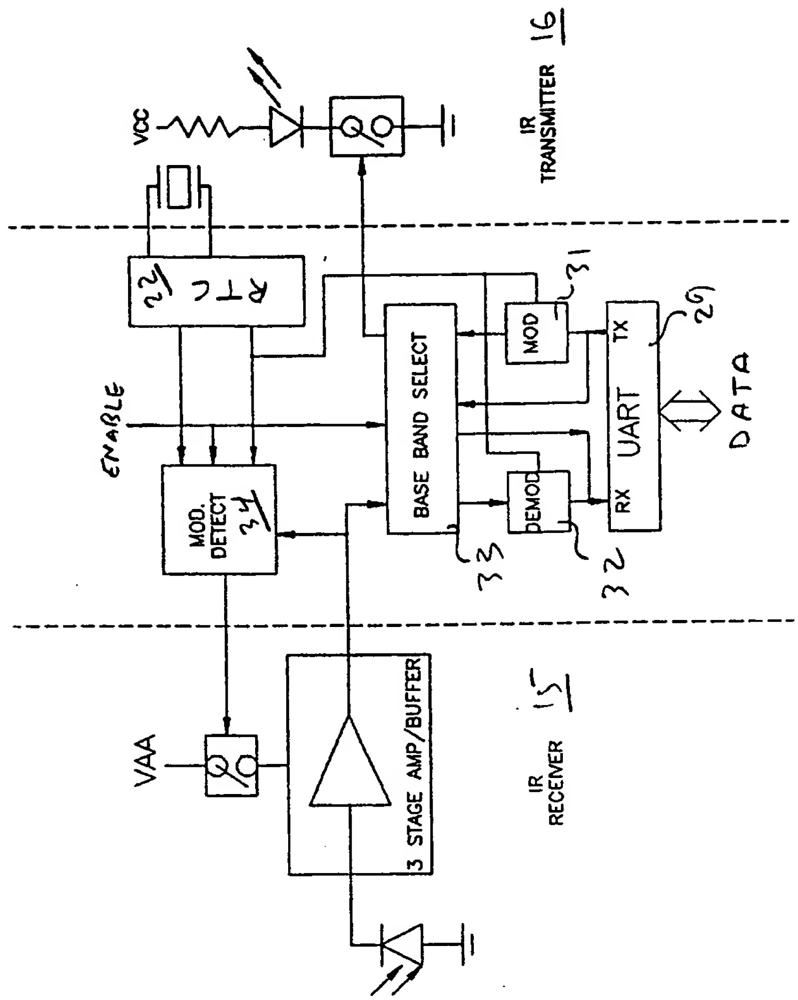
VCC
IR TRANSMITTER
16
RTC
22
MOD
31
TX
29
ENABLE
BASE BAND SELECT
UART
DATA
MOD. DETECT
34
DEMOD
RX
33
32
VAA
3 STAGE AMP/BUFFER
IR RECEIVER
15

FIGURE 3

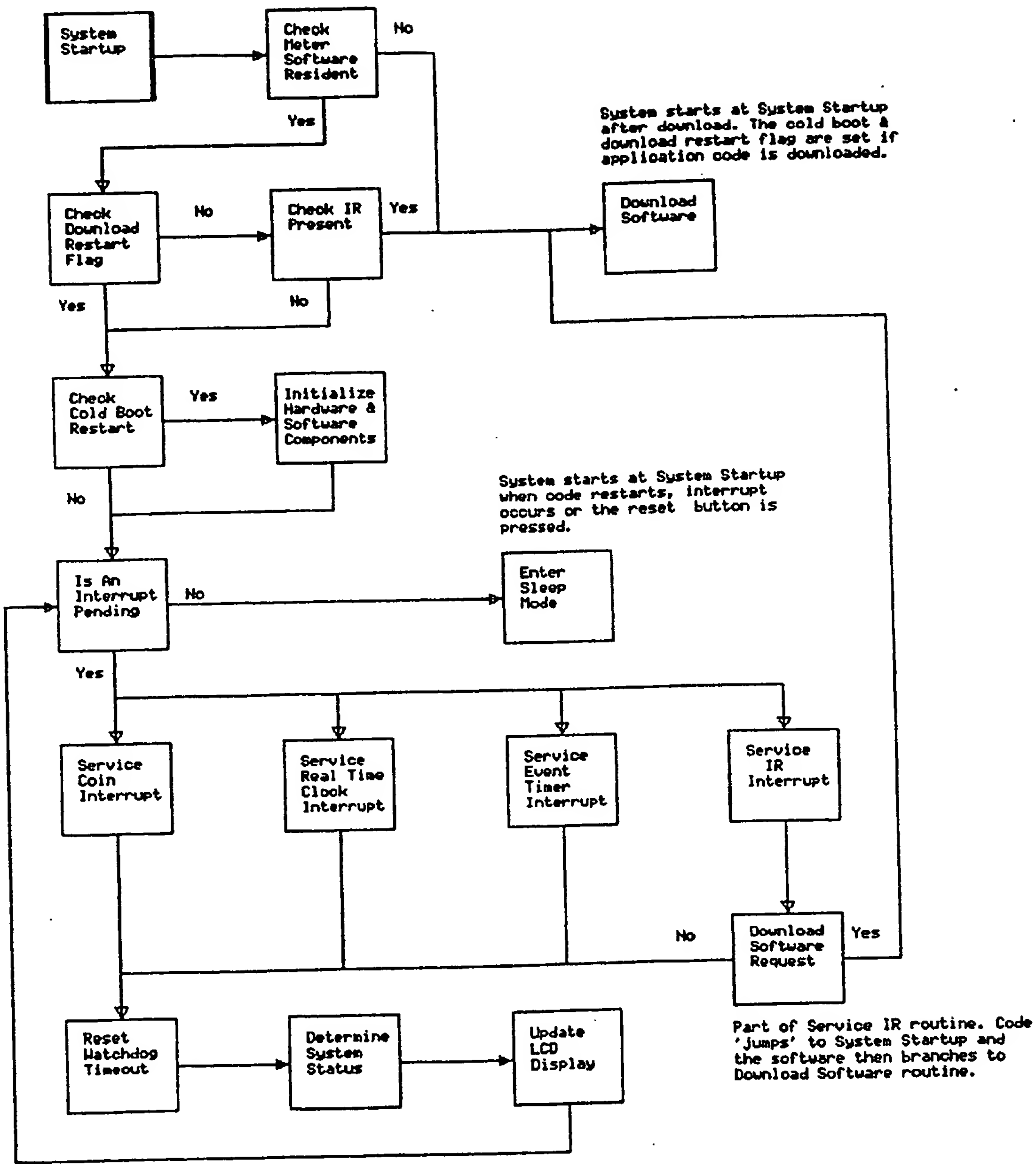
System Startup
Check Meter Software Resident
No
Yes
System starts at System Startup after download. The cold boot & download restart flag are set if application code is downloaded.
Check Download Restart Flag
No
Check IR Present
Yes
Download Software
Yes
No
Check Cold Boot Restart
Yes
Initialize Hardware & Software Components
No
System starts at System Startup when code restarts, interrupt occurs or the reset button is pressed.
Is An Interrupt Pending
No
Enter Sleep Mode
Yes
Service Coin Interrupt
Service Real Time Clock Interrupt
Service Event Timer Interrupt
Service IR Interrupt
No
Download Software Request
Yes
Part of Service IR routine. Code 'jumps' to System Startup and the software then branches to Download Software routine.
Reset Watchdog Timeout
Determine System Status
Update LCD Display

FIGURE 4

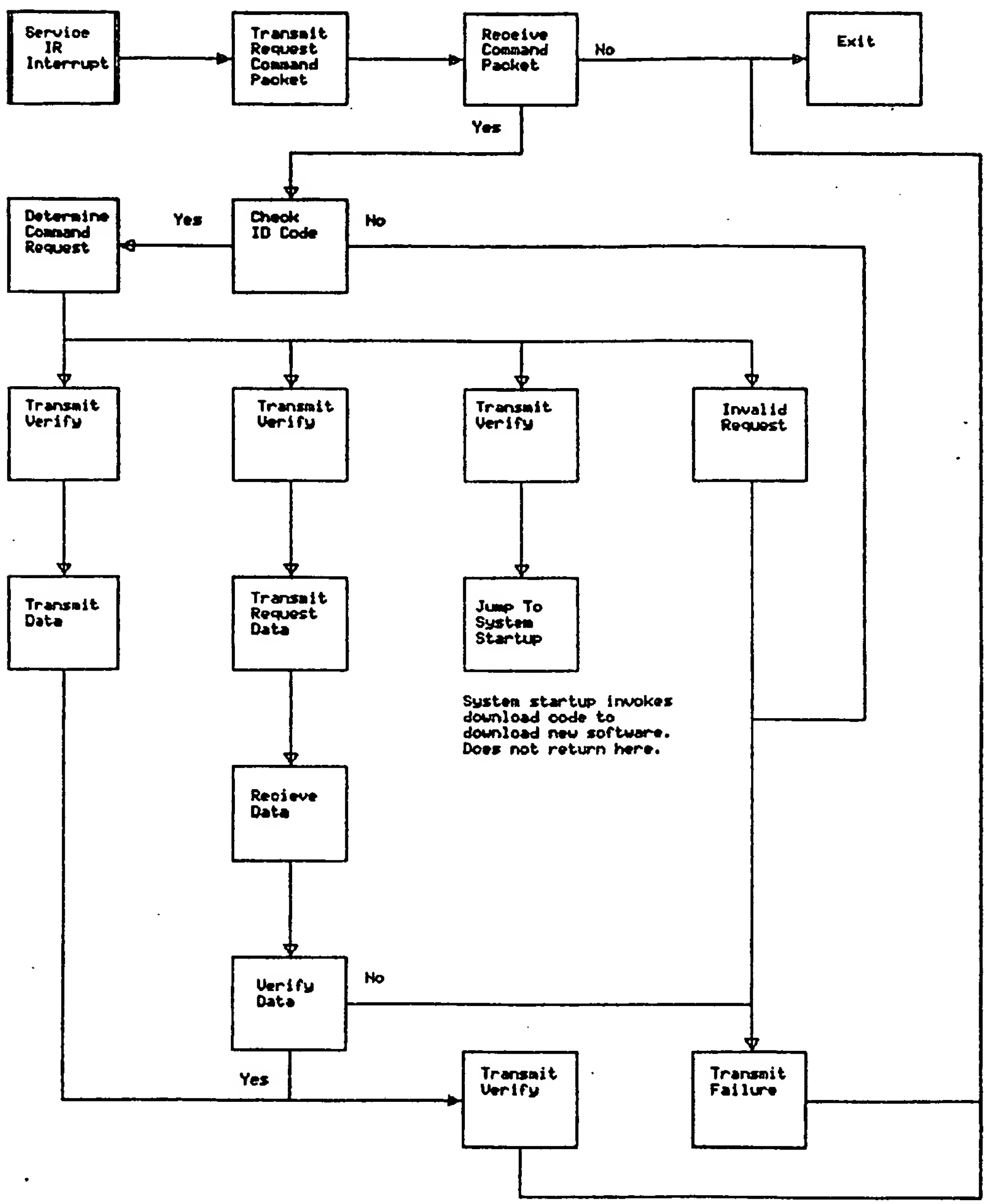
Service IR Interrupt
Transmit Request Command Packet
Receive Command Packet
No
Exit
Yes
Determine Command Request
Yes
Check ID Code
No
Transmit Verify
Transmit Verify
Transmit Verify
Invalid Request
Transmit Data
Transmit Request Data
Jump To System Startup
System startup invokes download code to download new software. Does not return here.
Recieve Data
Verify Data
No
Yes
Transmit Verify
Transmit Failure

FIGURE 5

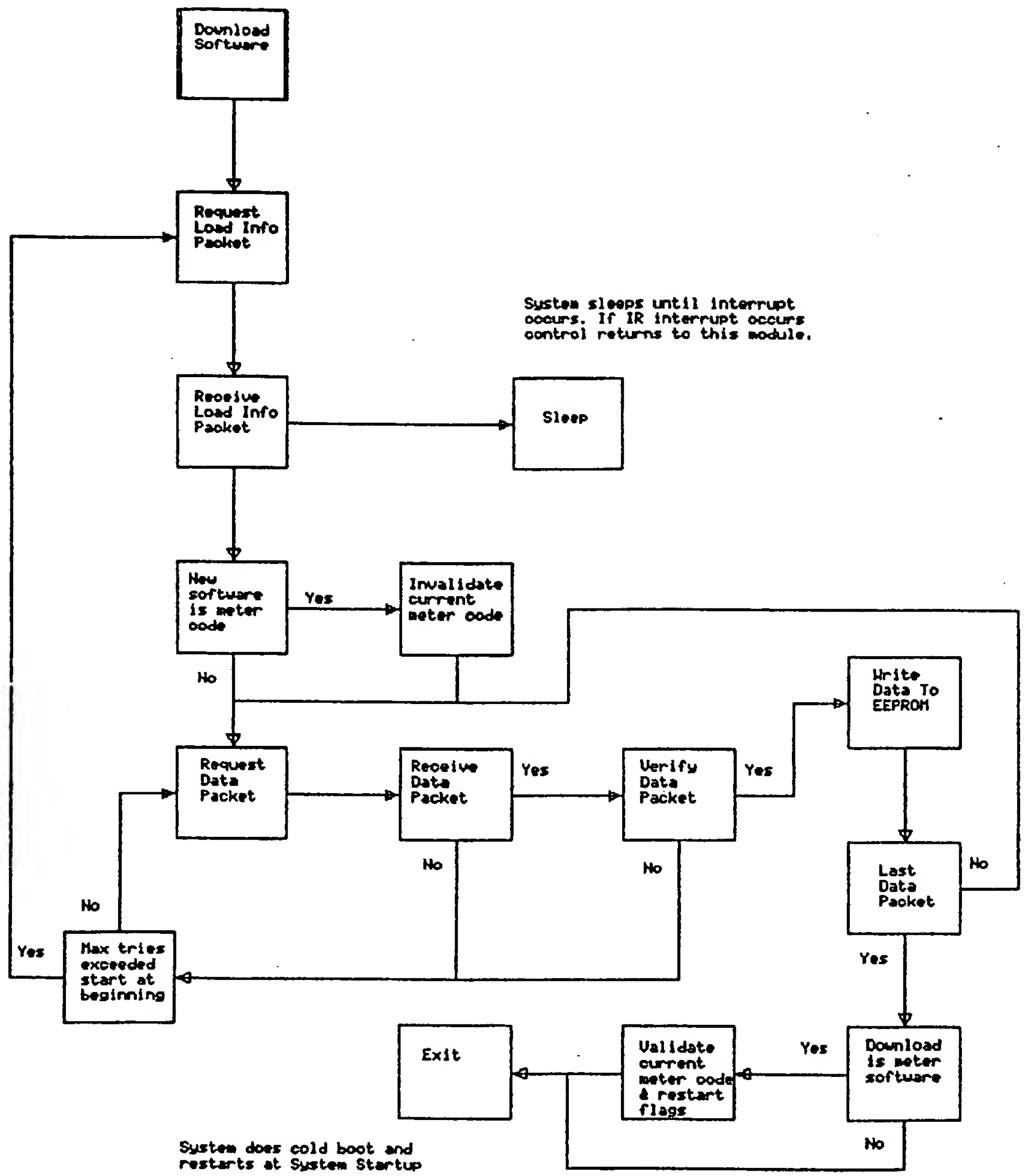
Download Software
Request Load Info Packet
System sleeps until interrupt occurs. If IR interrupt occurs control returns to this module.
Receive Load Info Packet
Sleep
New software is meter code
Yes
Invalidate current meter code
No
Write Data To EEPROM
Request Data Packet
Receive Data Packet
Yes
Verify Data Packet
Yes
No
No
Last Data Packet
No
No
Yes
Max tries exceeded start at beginning
Yes
Exit
Validate current meter code & restart flags
Yes
Download is meter software
No
System does cold boot and restarts at System Startup

FIGURE 6